**ATTORNEY DOCKET #APLE.P0011**
**Client Docket: 2585**

APPLICATION FOR UNITED STATES LETTERS PATENT

FOR

# Method and Apparatus for Incremental Code Signing

Inventors: Perry Keihtrieber
Michael Brouwer

Prepared by:
Stattler, Johansen & Adeli LLP
P.O. Box 51860
Palo Alto, California 94303-0728
PHONE: 650.752.0990 x101
FAX: 650.752.0995

**Method and Apparatus for Incremental Code Signing**

## BACKGROUND OF THE INVENTION

### Field of the Invention

The present invention relates to the field of computer security. In particular, the present invention discloses a system for verifying that an unauthorized party has not tampered with computer program code.

### Description of the Related Art

Computer security is one of the most pressing problems facing computer software makers. With the rise in popularity of the Internet, nearly every personal computer system is available on the Internet at one point or another. This great connectivity has provided many benefits to personal computer users. However, it has also provided a new host of problems to computer users. One of the biggest problems has been the rise of Internet transmitted viruses, worms, Trojan horses, and other malevolent programs.

Rogue computer programmers, sometimes known as "crackers", often attempt to break into computer systems to steal information or make unauthorized changes. Crackers use many different types of attacks in attempts to break into a computer system. Common methods employed by computer crackers include Trojan horses (a seemingly benign computer program that has a hidden agenda), a computer

virus (a piece of software that replicates itself from within a host program), a computer worm (a piece of software that replicates itself across a computer network), and social engineering (Deceiving a person with authorization codes into giving out those authorized codes).

These rogue computer programmers often alter existing legitimate programs by adding program code to perform unauthorized functions. By placing such authorized program code within legitimate programs, the rogue computer programmer thereby hides the unauthorized program code. The unauthorized code may thus dwell within a person's personal computer system without the person's knowledge for a long time. The unauthorized program code may destroy valuable data, waste computing resources (CPU cycles, network bandwidth, storage space, etc.), or pilfer confidential information.

In order to protect legitimate programs from such unauthorized adulteration, some software manufacturers generate a checksum of the program code. The checksum is a value calculated using the program code as input value such that each different computer program tends to have a different checksum value. The software manufacturer then digitally "signs" the checksum with a private key encryption key. Before running the computer program code, a user should then authenticate the program code. Specifically, the user has the personal computer system compute a checksum of the program code and then the computed checksum values is compared with the checksum calculated by the software manufacturer after

decrypting it with the software manufacturer's public key. If the two checksums match, then the computer program is presumed to be authentic.

These existing checksum based protection systems have been known to work relatively well. However, these existing checksum based protection systems can take a long time to authenticate large computer programs. Therefore, it would be desirable to provide a system faster system for authenticating large computer programs.

## SUMMARY OF THE INVENTION

The present invention discloses a method for quickly and easily authenticating large computer program. The system operates by first sealing the computer program with digital signature in an incremental manner. Specifically, the computer program is divided into a set of pages and a hash value is calculated for each page. The set of hash values is formed into a hash value array and then the hash value array is then sealed with a digital signature. The computer program is then distributed along with the hash value array and the digital signature. To authenticate the computer program, a recipient first verifies the authenticity of the hash value array with the digital signature and a public key. Once the hash value array has been authenticated, the recipient can then verify the authenticity of each page of the computer program by calculating a hash of a page to be loaded and then comparing with an associated hash value in the authenticated hash value array. If the hash values do not match, then execution may be halted.

Other objects, together with the foregoing are attained in the exercise of the invention described and illustrated in the accompanying embodiments and drawings.

## BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements, and in which:

**Figure 1** illustrates a conceptual diagram describing how a computer program can be sealed with a digital signature.

**Figure 2** illustrates a flow diagram that describes how a computer system verifies a computer program sealed with a digital signature before executing the computer program.

**Figure 3** illustrates a conceptual diagram describing how a computer program can be sealed with a digital signature using incremental code signing.

**Figure 4** illustrates a flow diagram that describes how a computer program can be sealed with a digital signature and a hash array using incremental code signing.

**Figure 5** illustrates a flow diagram describes how a computer system verifies and executes a computer program that has been digitally sealed with incremental code signing.

## DETAILED DESCRIPTION OF THE INVENTION

### Notation and Nomenclature

In the following detailed description of the present invention, numerous specific details are set forth in order to provide a thorough understanding of the present invention. However, it will become obvious to those skilled in the art that the present invention may be practiced without these specific details. In other instances, well known methods, procedures, components, and circuitry have not been described in detail to avoid unnecessarily obscuring aspects of the present invention.

The detailed description of the present invention in the following is presented largely in terms of procedures, steps, logic blocks, processing, and other symbolic representations that describe data processing devices coupled to networks. These process descriptions and representations are the means used by those experienced or skilled in the art to most effectively convey the substance of their work to others skilled in the art. The present invention is a method and apparatus for providing a mobile subscriber visual interface to customer care and billing systems. The method along with the apparatus, described in detail below, is a self-consistent sequence of processes or steps leading to a desired result. These steps or processes are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities may take the form of electrical signals capable of being stored, transferred, combined, compared, displayed and otherwise manipulated in computer systems or electronic computing devices. It proves convenient at times, principally for reasons of common usage, to refer to these signals as bits, values,

elements, symbols, operations, messages, terms, numbers, or the like. It should be borne in mind that all of these similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following description, it is appreciated that throughout the present invention, discussions utilizing terms such as "processing" or "computing" or "verifying" or "displaying" or the like, refer to the actions and processes of a computing device that manipulates and transforms data represented as physical quantities within the computing device's registers and memories into other data similarly represented as physical quantities within the computing device or other electronic devices.

**Computer Program Security with Digital Signatures**

To protect a computer program from unauthorized tampering, a software manufacturer may create a special "seal" for the computer program that should be tested before the program is executed. If the seal or the computer program code has been tampered with, then the test will fail such that the program will then refuse to execute.

A common method of creating such a seal is to use well-known "public key" encryption technology. Such a system that uses public key encryption technology will be described with reference to **Figures 1** and **2**.

Digitally Sealing a Computer Program

Referring now to the drawings, in which like numerals refer to like parts throughout the several views, **Figure 1** conceptually illustrates a process for creating a digital seal or digital signature for a computer program **100**. First, a hash is calculated for the entire computer program **100** using a hash function **110** to calculate a program hash value **120**. The program hash value **120**is a small representation derived from the computer program **100** such as a checksum. The program hash value **120** is then digitally signed with the private key **140** of a trusted entity using a digital signature function **130** to create a digital signature for the program hash **150**. The digital signature for the program hash **150** is the digital seal (or digital signature) that accompanies the program when it is distributed.

Verifying the Authenticity of a Digitally Sealed Computer Program

**Figure 2** illustrates a flow diagram that describes how the digital signature for the program hash **150** of **Figure 1** is used to verify a program's authenticity. Referring to **Figure 2**, a computer system using computer program **100** first loads the entire computer program **100** at step **210**. Then, in step **220**, the computer system calculates a program hash value for the entire computer program **100** (just as was done by the hash function **110** in **Figure 1**).

Next, in steps **230** and **240**, the computer system compares the digital signature for the program hash **150** that accompanied the computer program **100** with the calculated program hash value from step **220** using a well-known private key **235** of the trusted entity that created the digital signature. Specifically, the digital

signature for the program hash **150** is processed by the digital signature function with the public key **235** and the result may then be compared with the calculated program hash value from step **220**.

If the calculated hash function from step **220** matches the digital signature for the program hash **150** after being processed with the public key **235**, then the computer system proceeds to step **250** where program execution commences. However, if the calculated hash function from step **220** fails to match the digital signature for the program hash **150** after being processed with the public key **235**, then the computer system proceeds to step **260** and refuses to execute the computer program **100**.

The authentication system of **Figures 1** and **2** works satisfactorily for small programs, however it is ill suited for large computer programs. One significant problem with the authentication system of **Figures 1** and **2** can be found in steps **210** and **220** in the flow diagram of **Figure 2**. Steps **210** and **220** require that the entire computer program **100** be loaded into memory and then a hash value calculated across the entire computer program **100**. With a large program, this process can take a very long time. Thus, the user is forced wait for this entire time-consuming load and calculation process. Today's impatient computer users will simply not tolerate such long load times.

One particular large computer program that needs to be protected is the operating system for a computer system. The operating system has the authority to control all of the computers input/output devices such as disk drives, network

connections, displays, back-up systems, etc. Thus, it would be very desirable to have a computer program authentication system that could quickly and efficiently protect a computer operating system.

## Incremental Code Signing

The present invention introduces an improved method of digitally signing computer programs for security without introducing long load times present in existing systems. The present invention operates by breaking up the computer program into smaller units that are individually verified.

Digitally Sealing a Computer Program with Incremental Code Signing

**Figures 3** and **4** illustrate how the system of the present invention creates a digital seal for a program that can be used in a faster and more efficient manner. **Figure 3** presents a conceptual diagram of how the digital seal is created. **Figure 4** illustrates a detailed flow diagram that describes the method used to create the digital seal.

Referring to **Figures 3** and **4**, the present invention first divides the computer program **300** into a number of "pages" (**380** to **389**) in step **410**. Most computer systems already use a paged memory organization to implement a virtual memory system. Thus, the present invention can use the existing memory paging systems offered by a particular processor and/or operating system. Memory pages are typically 4 kilobytes ("k") or 8k in size.

Next, in steps **420** and **430**, the system calculates a hash value for each memory page of the computer program **300** using a hash function **310**. The hash function **310** may be any hash function such as the well-known SHA or MD5 has functions. As set forth in **Figure 3**, the hash function **310** will create an associated hash value (**390** to **389**) for each memory page (**380** to **389**) of the computer program **300**. The size of the output hash values in one embodiment are 20 bytes. However, many different sizes of hash values may be used.

In step **440**, the system of the present invention arranges the calculated hash values (**390** to **389**) into an array of hash values known as the hash array **373**. The system then calculates an array hash value **360** for the entire hash array **373** using a hash function **370** in step **450**. In one embodiment, hash function **370** is the same as hash function **310**. However, a different hash function may be used.

The trusted entity that is sealing the program then digitally signs the array hash value **360** with its private key **340** using a signature function **330** in step **460** to create a digital signature for the hash array **350**. Finally, at step **470**, the hash array **373** and the digital signature for the hash array **350** are stored along the computer program **300**. The hash array **373** and the digital signature for the hash array **350** are also distributed along with the computer program **300** such that any recipient of computer program **300** can verify its authenticity.

Verifying the Authenticity of a Digitally Sealed Computer Program

Once a hash array **373** and a digital signature for the hash array **350** have been created for a computer program, that computer program may be distributed

to user that may quickly and efficiently authenticate the computer program. **Figure 5** illustrates a flow diagram describes how the recipient of a computer program that has been digitally sealed with incremental code signing verifies and executes the digitally sealed computer program.

Referring to **Figure 5**, the recipient's personal computer system first loads the hash array that accompanies the computer program in step **510**. The computer system then calculates a hash value for the entire hash array at step **515**. Since the array of hash values is not very large, this particular hash computation may be completed very quickly.

Next, at steps **520** and **525**, the computer system then compares the calculated hash value with digital signature of the hash array that accompanied the computer program using the well-known public key of the trusted entity that sealed the computer program.

If the digital signature fails to match the hash value calculated from the hash array, then computer system proceeds to step **580** where it refuses the execute the computer program. Execution is refused since the digital signature and/or the hash array have been tampered with by an unauthorized entity.

Referring back to step **525**, if the digital signature matches the hash value calculated from the hash array then the computer system proceeds to step **530** where it loads a page of the computer program. The computer system then calculates a hash value for the loaded computer program page at step **540**. This calculation may

be performed within the memory paging mechanism of the computer's operating system. At steps **550** and **555**, the calculated hash value for the loaded computer program page is compared with hash value in the hash array that is associated with that particular memory page.

If the calculated hash value for the loaded computer program page does not match the associated hash value from the hash array, then the computer system proceeds to step **580** where it refuses to continue execution. This refusal to continue execution may be implemented as a page fault within the virtual memory system of the computer system. Other possible methods of signaling the failure could be to indicate that the page is not readable, there has been a verification error, or simply abort.

Referring back to step **555**, if the calculated hash value for the loaded computer program page matches the associated hash value from the hash array, then the computer system proceeds to step **560** where commences execution of the loaded page of the program. Eventually, the program will complete execution of the code in that loaded page (and previously loaded pages) and will either totally complete execution or need another page of the computer program, as set forth in step **570**. If the program is completely done, then the computer simply proceeds to step **590** where it is done.

If another page of the computer program is needed at step **570**, then the computer system proceeds back to step **530** to load the needed page of the computer

program. The newly loaded page will have to be authenticated by steps **540**, **550**, and **555**.

Note that the previously authenticated hash array is used by the computer system to further authenticate each memory page that is loaded. Thus, computer must ensure that the authenticated hash array is not tampered with during program execution. For example, if the operating system swaps the authenticated hash array out of protected memory, the hash array should be re-authenticated once it is loaded back in to ensure its authenticity.

This written specification and the accompanying drawings have disclosed the present invention in sufficient detail with a certain degree of particularity. It is understood to those skilled in the art that the present disclosure of embodiments has been made by way of example only and that numerous changes in the arrangement and combination of parts as well as steps may be resorted to without departing from the spirit and scope of the invention as claimed. Accordingly, the scope of the present invention is defined by the appended claims rather than the forgoing description of embodiments.